EXHIBIT J

Real-Time Universal Capture Facial Animation with GPU Skin Rendering

Meng Yang
mengyang@seas.upenn.edu

PROJECT ABSTRACT

The project implements the real-time skin rendering algorithm presented in [1], and further integrates the rendering technique with Universal Capture (UCap) [2] facial animation. The rendering approach approximates the appearance of subsurface scattering by blurring the diffuse illumination in texture space using graphics hardware. This approach, based on the offline skin rendering technique proposed by Borshukov and Lewis [3], gives a realistic look and is both efficient and straightforward to implement.

1. INTROUDUCTION

The ultimate challenge in photorealistic computer graphics is rendering believable human faces. We are trained to study the human face since birth, so our brains are intimately familiar with every nuance and detail of what human skin is supposed look like. The challenge of rendering human skin is further complicated by some technical issues such as the fact that most lighting from skin comes from sub-surface scattering.

Universal Capture is a facial animation capture technique developed first for The Matrix Reloaded and The Matrix Revolutions and then brought to Electronics Arts where it has been adapted to real-time use. Universal Capture has been or will soon be featured in trailers for Fight Night and in-game in Tiger Woods and Need For Speed Carbon. Universal Capture combines the accuracy of traditional facial motion capture techniques with the accuracy of multi-angle high definition video to produce digital versions of an actor that move and look just like the real person.

Universal Capture has produced some of the highest quality captured facial animation to date in film or games and looks set to continue doing so. Universal Capture provides the ability to record and replay an individual's facial movements and speech with high fidelity. Current development efforts are focused on sequencing together and looping through different captured clips.

1.1. Technology

The project is developed primarily on nVidia GeForce 6800 card in Moore 100B. The shaders are written in CG, controlled by external program written in C^{++}. Mel scripting language is used to export UCap data in a desirable format suitable for this project. The full reference list is appended to the end of the document.

2. PROJECT DEVELOPMENT APPROACH

2.1. Algorithm Details

For each frame, the real-time rendering algorithm proceeds as follows:

1. Render diffuse illumination to 2D map from camera's view
2. Render from light's point of view for depth map
3. Use the depth map to compute shadow map
4. Update diffuse illumination with shadows
5. Blur the light map using Poisson distribution model
6. Load UCap texture for the current frame
7. Render final mesh using the blurred 2D light map for diffuse illumination

The above steps are straightforward to implement using pixel shaders and renderable textures. We will describe how to compute shadows and how to perform the blur operation. Finally, we will describe additional acceleration techniques.

2.1.1 Render-passes Control

We utilize FBO (Frame Buffer Object) as intermediate off-screen renderable textures to store temporary results from each pass. Each pass takes the target and source textures as input parameters, handles OpenGL matrix transformations, turns on the corresponding CG programs, passes the CG parameters and performs the drawing. Different combinations of the modularized passes allow users to real-time toggle and view the results between different render states.

2.1.2 Soft Shadows

Using this texture-space skin rendering technique, the computation of soft shadows is relatively inexpensive. A shadow map algorithm is used to determine visibility from the light, and texels in the diffuse light map are dimmed accordingly. The blur operation performed in step 3 will not only create the appearance of subsurface scattering, it will also provide soft shadows at no additional cost. The blur pass also significantly reduces aliasing, making it practical to use the shadow map algorithm.

Since it is fairly expensive to perform an independent blur pass for the shadow component, we cannot independently apply shadows to the specular illumination. We

have found, however, that the luminance of the blurred light map can be used to attenuate the specular term of the shadow casting light to obtain a natural look.

2.1.3 The Poisson Disc Blur

We implemented the blur operation in hardware by using a pixel shader that applies a variable sized Poisson disc filter to the character's skin light maps in texture space. Using two temporary buffers, we performed several blurring passes on the diffuse illumination in order to achieve a soft, realistic look. [4]

Variable kernel size. The kernel size used by our filter can vary in texture space. A scalar multiplier for the kernel size is stored in an 8-bit texture. Borshukov and Lewis addressed translucency on the model's ears by ray-tracing. We cannot afford to do ray-tracing in real-time, but we obtain a similar result by increasing the kernel size on the region around the ears.

Texture boundary dilation. In order to prevent boundary artifacts when fetching from the light map, the texture needs to be dilated prior to blurring. We needed an efficient real-time solution to this problem. We accomplished this by modifying the Poisson disc filter shader to check whether a given sample is just outside the boundary of useful data, and if so, copy from an interior neighboring sample instead. We only need to use this modified, more expensive filter in the first blurring pass.

2.1.4 UCap Facial Animation Data

Universal Capture animation data is stored as animation curves for each key joint that drives the face mesh. For each joint, there's weight map indicating how much influence it has on its neighboring vertices. We process the animation data as follows:

- Export per-joint UCap animation curves and per-joint weight map through Maya's Mel scripting language
- Import the exported data into our application
- For each frame
    - Loop through all the joints to update weighted positions of vertices
    - Loop through all triangles to update per-vertex normals
    - UVs remain the same
    - Load UCap texture of the current frame

The above procedure works efficiently for large-scale animations with bulk size of animation data at the cost of reduced frame rate (FPS). For short animations, an alternative method is to export per-frame positions and normals for all vertices with Mel script and preload the data during the application's initialization. This method produces favorable FPS but requires relatively long loading time. Our application implements the 2nd method.

Other feasible applications of this rendering algorithm include customized hardware shading node plugin for Maya, which allows users to edit UCap data and immediately view the real-time results.

2.1.5 Acceleration Techniques

In order to optimize our technique, we employ hardware early-z culling to avoid processing certain regions of the light map.

Frustum culling. Before the blurring step, we clear the z buffer to 1 and perform a very simple and cheap texture space rendering pass in which we just set the z value to 0 for all rendered samples. If the bounding box of the model's head lies outside the view frustum, it is culled by our graphics engine, and thus the z value is not modified. On all further texture-space passes, we set the z value to 0 and the z test to "equal". This ensures that if the model lies outside the view frustum, hardware early-z culling will prevent all samples from being processed. [5]

Backface culling. Backface culling can be performed by first computing the dot product of the view vector and normal. Then, if the sample is frontfacing, a 0 is written to the z buffer, thus culling the sample in all further passes.

Distance culling. If the model lies far from the camera, the z value is set to 1, and the light map from the previously rendered frame is used. Note that this does not affect specular lighting.

3. RESULTS

Our application needs approximately 3 minutes to load a 150-frame UCap animation and renders the interactive scene under steady per second frame rate around 18.0 (with all five render passes turned on) or 32.0 (without the shadow mapping), tested on NVIDIA's 6800 graphic card. The loading time can be decently reduced by other approach discussed above to load the UCap data. The following images are all directly captures from our application where you can interactively trigger between different render passes to view the results.

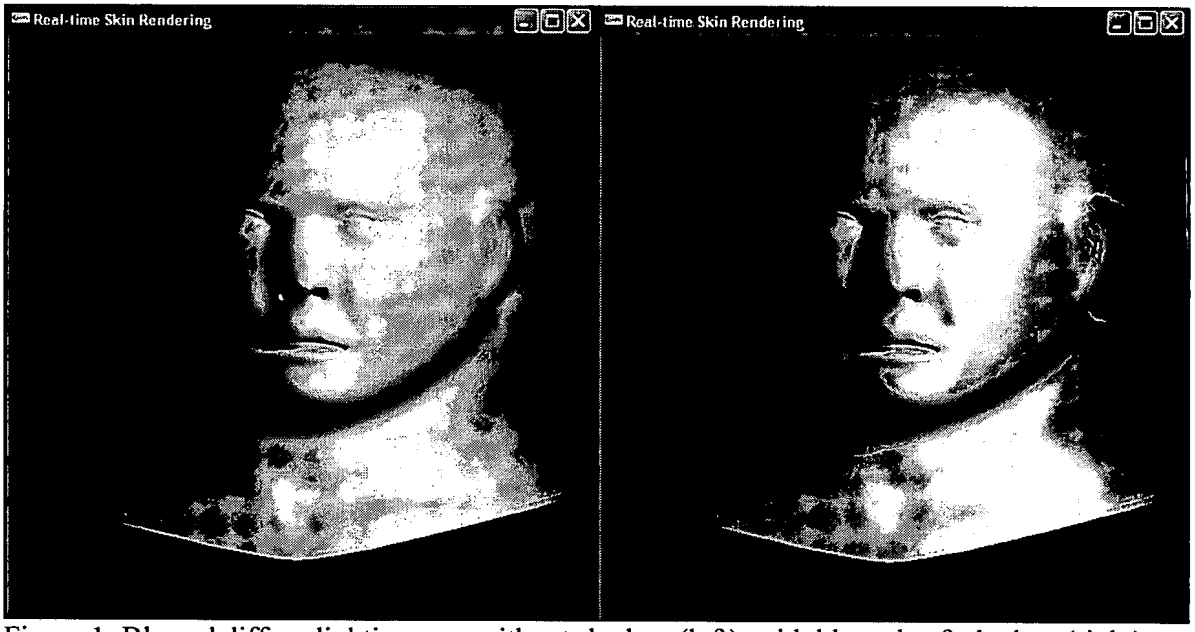

Figure 1: Blurred diffuse lighting map without shadow (left); with blurred soft shadow (right).

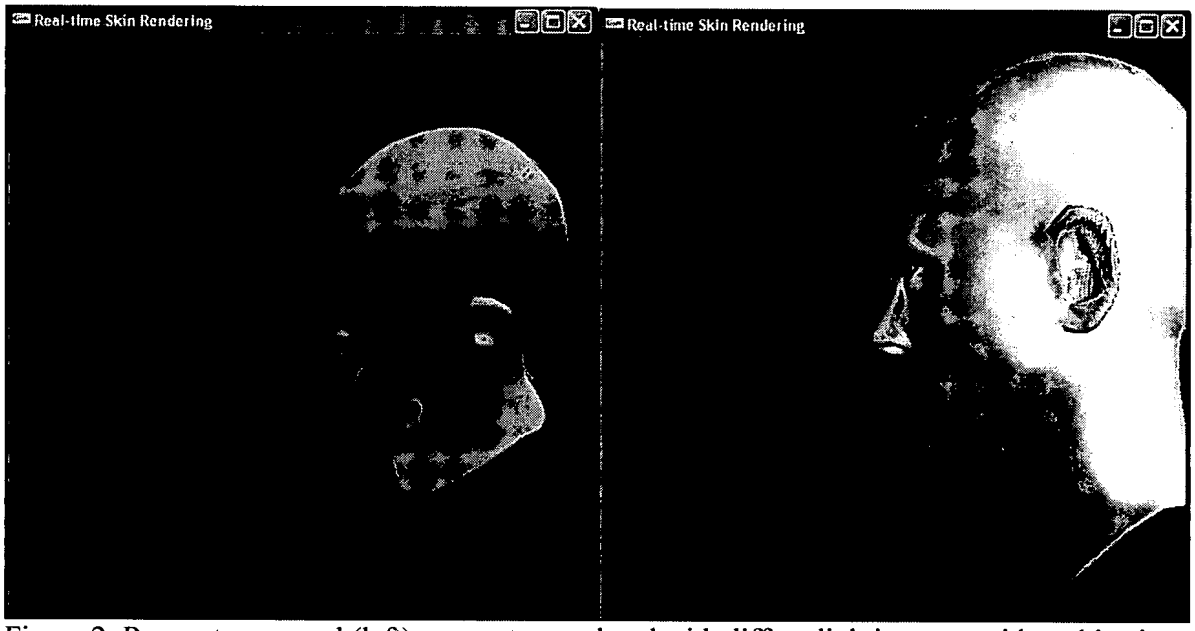

Figure 2: Per-vertex normal (left); geometry rendered with diffuse lighting map with no blurring

Figure 3: Captured animation sequences with full skin rendering turned on

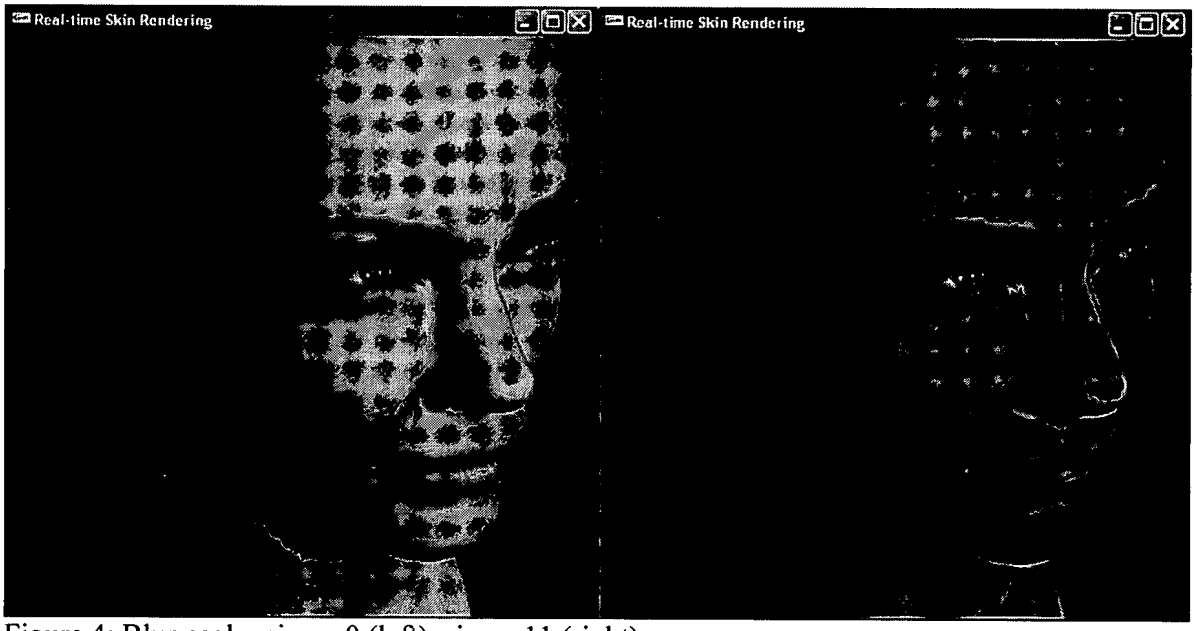

Figure 4: Blur-scaler size = 0 (left); size = 11 (right)

Figure 4: Rendered with blur-scaler size = 11

4. CONCLUSION AND FUTURE WORK

Rendering UCap animation in real-time, combined with advanced GPU shading techniques can achieve realistic and engaging visual experience. The current shadow mapping algorithm implemented in this application is heavily dependant on the blur-scaler lookup map. As a result we should search for better and more efficient shadow mapping algorithms, as well as other approach to approximate the subsurface scattering, such as the Enhanced PRT Based Lighting from ATI's Ruby 2 demo.

Even though UCap and MoCap are similar in nature, UCap is not skeleton-based animation but rather keyframe-based. And hence the UCap animation is associated with a specific geometry and not applicable to other models with different geometric structure. Another direction for future work is to explore algorithms that perform fast UCap skin geometry deforming on GPU, in order to produce highly stylized facial animation in real-time.

5. REFERENCES

[1] PEDRO S., DAVID G., AND JASON M. 2004. Real-Time Skin Rendering on Graphics Hardware. *Sketches, SIGGRAPH 2004.*

[2] BORSHUKOV G., MONTGOMERY J. AND WERNER W. 2006. Playable Universal Capture: Compression and Real-time Sequencing of Image-Based Facial Animation. *Courses, SIGGRAPH 2006.*

[3] BORSHUKOV G. AND LEWIS, J. P. 2003. Realistic Human Face Rendering for "The Matrix Reloaded". *Sketches, SIGGRAPH 2003.*

[4] OAT C. 2006. Rendering Gooey Materials with Multiple Layers. *Courses 26 SIGGRAPH 2006.*

[5] Mitchell J. AND SANDER P. 2004. Applications of Explicit Early-Z Culling. *Real-Time Shading Course, SIGGRAPH 2004*